# CS 124 Programming Assignment 2: Spring 2022

**Your name(s) (up to two):** Burak Ufuktepe

**Collaborators:** None

No. of late days used on previous psets: 10

No. of late days used after including this pset:

## Analytical Approach

Let and be the time required to multiply two matrices using the conventional matrix multiplication algorithm and Strassen’s algorithm, respectively.

Let be the time required to multiply two matrices using the conventional matrix multiplication algorithm. The conventional matrix multiplication requires multiplications and additions for each element and there are elements in the matrix. Therefore, the runtime for the conventional matrix multiplication is:

Let be the time required to multiply two matrices using Strassen’s algorithm. Strassen’s algorithm performs 7 multiplications and 18 additions/subtractions of matrices of size . The reason why we use a ceiling function is because needs to be divided in half evenly, if is odd we would need to pad the matrix with one extra zero row and one extra zero column. Thus, we can express the recurrence relation as follows:

To find the cross-over point we need to find the value for when . So, we can simply use the equality and solve for .

If is even, we have:

This shows when is even and less than 15, switching to the conventional algorithm will be more optimal than using Strassen’s algorithm.

If is odd, we will have to pad the matrices with one extra zero row and one extra zero column. Let . After padding the matrices and dividing by half we obtain:

So we have:

This shows when is odd and less than 37.2, switching to the conventional algorithm will be more optimal than using Strassen’s algorithm.

## Implementation

### Data Layout Optimization

Splitting matrices takes up a significant portion of the actual runtime in Strassen’s algorithm. To speed up this process, instead of using a standard row-major ordering, Morton ordering is used to represent matrices. Morton ordering takes a 2D array stored in row-major order and arranges the matrix in block arrays where is the size of the “base case”. This is illustrated in Figure 1 for and Figure 2 for .

Table

Description automatically generated

Figure 1: Row-Major Ordering vs Morton Ordering (2 x 2 blocks)

Diagram

Description automatically generated with medium confidence

Figure 2: Row-Major Ordering vs Morton Ordering (3 x 3 blocks)

This way we end up with a 1D array and each quadrant is stored contiguously in memory as shown in Figure 3. To partition an matrix into four matrices we can simply slice the 1D array into four parts without having to iterate over each element.

Text

Description automatically generated with low confidence

Figure 3: 1D Array Representation of a Matrix using Morton Ordering

To compare the performance of Morton Ordering to Row-Major Ordering, several trials are conducted using random matrices of size . Cross-over points are chosen between 10 and 100 and runtimes are plotted for each cross-over point as illustrated in Figure 4.

Morton Ordering shows a noticeable perfomance gain over Row-Major Ordering (7% to 26%). The largest performance gain is observed when the cross-over point is small. This is because using a smaller cross-over point results in more recursive calls in Strassen’s algorithm, as a result the number of matrix partition operations increases. And splitting up a matrix in Morton Ordering is a lot cheaper compared to Row-Major Ordering.

Chart, line chart

Description automatically generated

Figure 4: Morton Ordering vs Row-Major Ordering Runtime Comparison

### Conventional Matrix Multiplication Optimization

### Padding

Strassen’s algorithm recursively divides matrices into four matrices. That means at each recursive call must be divisible by 2. The obvious method to resolve this issue would be to pad the original matrix to the next power of two with zero rows and zero columns. However, this would be an expensive approach because we would have to almost double the size of the original matrix if its size is just over a power of 2 (i.e ).

Instead, we use the following approach: we first find the size of the “base case” for our Morton-ordered array. Then we keep doubling that size until we reach or exceed the size of our original matrix to find the minimum required size that we need to perform Strassen’s algorithm. That is:

1. Let . Repeatedly divide in half, each time taking the ceiling, until is less than or equal to the cross-over point. This will be equal to the size of the “base case” for our Morton-ordered array.
2. Then repeatedly double until .
3. Pad the original matrix with zero rows and zero columns until we obtain a matrix of size .

In other words, we are padding the matrix just enough so that Strassen’s algorithm can reach the “base case”.

For example, when and the cross-over point is 37, we would first find the size of the base case as follows:

🡪 🡪 🡪 🡪

Then we can repeatedly double this size until we reach or exceed to find the minimum required matrix size:

🡪 🡪 🡪 🡪